

## Assignment 3 - Part A

Generated by Doxygen 1.12.0



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 BinControl Class Reference	5
3.1.1 Detailed Description	5
3.1.2 Constructor & Destructor Documentation	5
3.1.2.1 BinControl()	5
3.1.3 Member Function Documentation	6
3.1.3.1 binCheck()	6
3.1.3.2 compressFile()	6
3.1.3.3 erase()	6
3.1.3.4 get()	7
3.1.3.5 insert()	7
3.2 LRUCacheElement Class Reference	8
3.2.1 Detailed Description	8
3.2.2 Constructor & Destructor Documentation	8
3.2.2.1 LRUCacheElement()	8
3.3 LRUCacheSegment Class Reference	9
3.3.1 Detailed Description	9
3.3.2 Constructor & Destructor Documentation	9
3.3.2.1 LRUCacheSegment()	9
3.3.3 Member Function Documentation	9
3.3.3.1 get()	9
3.3.3.2 put()	10
3.3.3.3 remove()	10
3.4 PersistentHashMap Class Reference	11
3.4.1 Detailed Description	11
3.4.2 Member Enumeration Documentation	11
3.4.2.1 INITIALIZATION_MODE	11
3.4.3 Constructor & Destructor Documentation	12
3.4.3.1 PersistentHashMap()	12
3.4.3.2 ~PersistentHashMap()	13
3.4.4 Member Function Documentation	13
3.4.4.1 erase()	13
3.4.4.2 get()	13
3.4.4.3 insert()	14
3.5 SpinLock Class Reference	14
3.5.1 Detailed Description	14
3.6 StorageEngine Class Reference	15

3.6.1 Detailed Description . . . . .	15
3.6.2 Member Enumeration Documentation . . . . .	15
3.6.2.1 INITIALIZATION_MODE . . . . .	15
3.6.3 Constructor & Destructor Documentation . . . . .	15
3.6.3.1 StorageEngine() . . . . .	15
3.6.4 Member Function Documentation . . . . .	16
3.6.4.1 erase() . . . . .	16
3.6.4.2 get() . . . . .	16
3.6.4.3 insert() . . . . .	16
<b>4 File Documentation</b>	<b>19</b>
4.1 lru_cache.h . . . . .	19
4.2 persistent_hashmap.h . . . . .	19
4.3 storage_engine.h . . . . .	20
<b>Index</b>	<b>23</b>

# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">BinControl</a>	A class to represent a bin in the persistent hash map . . . . .	5
<a href="#">LRUCacheElement</a>	A class to represent an element in the LRU cache . . . . .	8
<a href="#">LRUCacheSegment</a>	A class to represent a segment in the LRU cache . . . . .	9
<a href="#">PersistentHashMap</a>	A class to represent a disk-persistent hash map . . . . .	11
<a href="#">SpinLock</a>	A simple spin lock implementation using atomic_flag . . . . .	14
<a href="#">StorageEngine</a>	A class to represent a disk-persistent key-value store . . . . .	15



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

include/ <a href="#">lru_cache.h</a> . . . . .	19
include/ <a href="#">persistent_hashmap.h</a> . . . . .	19
include/ <a href="#">storage_engine.h</a> . . . . .	20





# Chapter 3

## Class Documentation

### 3.1 BinControl Class Reference

A class to represent a bin in the persistent hash map.

```
#include <persistent_hashmap.h>
```

#### Public Member Functions

- [BinControl](#) (int bin\_id, std::string bin\_path, int cache\_capacity)  
*Constructs a new [BinControl](#) object.*
- void [insert](#) (std::string key, std::string value)  
*Inserts a key-value pair into the bin.*
- bool [erase](#) (std::string key)  
*Erases a key-value pair from the bin.*
- std::optional< std::string > [get](#) (std::string key)  
*Retrieves the value of a key from the bin.*
- void [compressFile](#) ()  
*Compresses the bin file.*
- void [binCheck](#) ()  
*Checks and fixes the bin file.*

#### 3.1.1 Detailed Description

A class to represent a bin in the persistent hash map.

This class provides a class to represent a bin in the persistent hash map. A bin is a file that stores key-value pairs. The class provides methods to insert, erase, and get key-value pairs from the bin. It also provides methods to compress the file and check/fix the file.

#### 3.1.2 Constructor & Destructor Documentation

##### 3.1.2.1 BinControl()

```
BinControl::BinControl (  
    int bin_id,  
    std::string bin_path,  
    int cache_capacity)
```

Constructs a new [BinControl](#) object.

## Parameters

<i>bin_id</i>	The ID of the bin.
<i>bin_path</i>	The path of the bin.
<i>cache_capacity</i>	The capacity of the LRU cache segment.

### 3.1.3 Member Function Documentation

#### 3.1.3.1 binCheck()

```
void BinControl::binCheck ()
```

Checks and fixes the bin file.

Checks the bin file for consistency. Reads entries and checks if they are valid. Invalid entries, which may occur towards the end of the file, are removed.

This method checks the bin file for any inconsistencies and fixes them if found. Writer lock to write to the file, allowing only one writer at a time. Ensures no other thread accesses the file while it is being checked.

Iterating through the file to read each entry and check if it is valid

Checking if the entry fits within the file size

#### 3.1.3.2 compressFile()

```
void BinControl::compressFile ()
```

Compresses the bin file.

This method compresses the bin file by removing any deleted key-value pairs and reorganizing the file accordingly. Writer lock to write to the file, allowing only one writer at a time. Ensures no other thread accesses the file while it is being compressed.

< The current read position in the file

< The current write position in the file

Iterating through the file to read each entry, and writing it back to the file if it is not marked as deleted

If the entry is marked as deleted, skip it

Writing the non-deleted entry back to the file

Updating the write position to the end of the written entry

Truncating the file to the final write position

#### 3.1.3.3 erase()

```
bool BinControl::erase (
    std::string key)
```

Erases a key-value pair from the bin.

**Parameters**

<i>key</i>	The key of the key-value pair to erase.
------------	---

**Returns**

True if the key-value pair was erased, false otherwise.

Writer lock to write to the cache and the file, allowing only one writer at a time

Remove the key from the cache

Iterating through the file to find the key, and if found, mark it as deleted and return true

Skip this entry if it is marked as deleted or if lengths mismatch

If the key is found, mark it as deleted and return true

**3.1.3.4 get()**

```
std::optional< std::string > BinControl::get (  
    std::string key)
```

Retrieves the value of a key from the bin.

**Parameters**

<i>key</i>	The key to retrieve the value for.
------------	------------------------------------

**Returns**

The value of the key, if it exists.

Reader lock to read from the cache and the file, allowing multiple readers at a time

If the key is found in the cache, return the value

Iterating through the file to find the key, and if found, updating the cache and returning the value

Skip this entry if it is marked as deleted or if lengths mismatch

If the key is found, read the value, update the cache, and return the value

**3.1.3.5 insert()**

```
void BinControl::insert (  
    std::string key,  
    std::string value)
```

Inserts a key-value pair into the bin.

**Parameters**

<i>key</i>	The key of the key-value pair.
<i>value</i>	The value of the key-value pair.

Ensure that only one thread can write to the bin at a time. This includes writing to the cache and the file.

Iterating through the file to find the key if it exists, and mark it as deleted if it does

Storing the starting position of the current entry, to be used for marking the entry as deleted

Skip this entry if it is marked as deleted or if lengths mismatch, as it is not the key we are looking for

If the key is found, mark it as deleted

We don't need to read the value, so we skip it

Write the new key-value pair to the end of the file

The documentation for this class was generated from the following files:

- include/persistent\_hashmap.h
- src/persistent\_hashmap.cpp

## 3.2 LRUCacheElement Class Reference

A class to represent an element in the LRU cache.

```
#include <lru_cache.h>
```

**Public Member Functions**

- [LRUCacheElement](#) (std::string [key](#), std::string [value](#))  
*Constructs a new [LRUCacheElement](#) object.*

**Public Attributes**

- std::string [key](#) {}  
*The key of the element.*
- std::string [value](#) {}  
*The value of the element.*

### 3.2.1 Detailed Description

A class to represent an element in the LRU cache.

This class provides a class to represent an element in the LRU cache.

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 LRUCacheElement()

```
LRUCacheElement::LRUCacheElement (
    std::string key,
    std::string value)
```

Constructs a new [LRUCacheElement](#) object.

## Parameters

<i>key</i>	The key of the element.
<i>value</i>	The value of the element.

The documentation for this class was generated from the following files:

- include/lru\_cache.h
- src/lru\_cache.cpp

## 3.3 LRUCacheSegment Class Reference

A class to represent a segment in the LRU cache.

```
#include <lru_cache.h>
```

## Public Member Functions

- [LRUCacheSegment](#) (int capacity)  
*Constructs a new [LRUCacheSegment](#) object.*
- void [put](#) (std::string key, std::string value)  
*Inserts a new element into the LRU cache segment.*
- std::optional< std::string > [get](#) (std::string key)  
*Retrieves the value of an element from the LRU cache segment.*
- void [remove](#) (std::string key)  
*Removes an element from the LRU cache segment.*

### 3.3.1 Detailed Description

A class to represent a segment in the LRU cache.

This class provides a class to represent a segment in the LRU cache.

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 LRUCacheSegment()

```
LRUCacheSegment::LRUCacheSegment (  
    int capacity)
```

Constructs a new [LRUCacheSegment](#) object.

## Parameters

<i>capacity</i>	The capacity of the LRU cache segment.
-----------------	--

### 3.3.3 Member Function Documentation

#### 3.3.3.1 get()

```
std::optional< std::string > LRUCacheSegment::get (  
    std::string key)
```

Retrieves the value of an element from the LRU cache segment.

**Parameters**

<i>key</i>	The key of the element.
------------	-------------------------

**Returns**

The value of the element, if it exists.

Ensure that only one thread can access the cache segment at a time

Check if the key exists in the cache segment. If it does, move the element to the front of the list. Otherwise, return an empty optional.

**3.3.3.2 put()**

```
void LRUCacheSegment::put (
    std::string key,
    std::string value)
```

Inserts a new element into the LRU cache segment.

**Parameters**

<i>key</i>	The key of the element.
<i>value</i>	The value of the element.

Ensure that only one thread can access the cache segment at a time

Check if the key already exists in the cache segment. If it does, update the value and move the element to the front of the list. Otherwise, insert a new element at the front of the list and evict the least recently used element if the capacity is exceeded.

Update existing element

Insert new element

Evict least recently used element if capacity is exceeded

**3.3.3.3 remove()**

```
void LRUCacheSegment::remove (
    std::string key)
```

Removes an element from the LRU cache segment.

**Parameters**

<i>key</i>	The key of the element.
------------	-------------------------

Ensure that only one thread can access the cache segment at a time

Check if the key exists in the cache segment. If it does, remove the element from the list and map.

The documentation for this class was generated from the following files:

- include/lru\_cache.h
- src/lru\_cache.cpp

## 3.4 PersistentHashMap Class Reference

A class to represent a disk-persistent hash map.

```
#include <persistent_hashmap.h>
```

### Public Types

- enum class `INITIALIZATION_MODE` { `CREATE` = 0 , `OPEN` = 1 }  
*An enumeration to specify the initialization mode of the persistent hash map.*

### Public Member Functions

- `PersistentHashMap` (std::string directory, int num\_bins, `INITIALIZATION_MODE` mode=`INITIALIZATION_MODE::CREATE`)  
*Constructs a new `PersistentHashMap` object.*
- `~PersistentHashMap` ()  
*Destroys the `PersistentHashMap` object.*
- void `insert` (std::string key, std::string value)  
*Inserts a key-value pair into the hash map.*
- bool `erase` (std::string key)  
*Erases a key-value pair from the hash map.*
- std::optional< std::string > `get` (std::string key)  
*Retrieves the value of a key from the hash map.*

### 3.4.1 Detailed Description

A class to represent a disk-persistent hash map.

This class provides a class to represent a persistent hash map. The persistent hash map stores key-value pairs in bins, which are files that store the key-value pairs. The class provides methods to insert, erase, and get key-value pairs from the hash map. It also provides a garbage collector to compress the bins.

### 3.4.2 Member Enumeration Documentation

#### 3.4.2.1 INITIALIZATION\_MODE

```
enum class PersistentHashMap::INITIALIZATION_MODE [strong]
```

An enumeration to specify the initialization mode of the persistent hash map.

This enumeration specifies the initialization mode of the persistent hash map. The persistent hash map can be created or opened.

#### Enumerator

CREATE	Create a new persistent hash map.
OPEN	Opens an existing persistent hash map.

### 3.4.3 Constructor & Destructor Documentation

#### 3.4.3.1 PersistentHashMap()

```
PersistentHashMap::PersistentHashMap (  
    std::string directory,  
    int num_bins,  
    INITIALIZATION_MODE mode = INITIALIZATION_MODE::CREATE)
```

Constructs a new [PersistentHashMap](#) object.



## Parameters

<i>directory</i>	The directory where the bins are stored.
<i>num_bins</i>	The number of bins in the hash map.
<i>mode</i>	The initialization mode of the hash map.

Creating bin controls for each bin

If the mode is CREATE, clear the directory and create new bucket files. Overwrite existing files if they exist.

If the mode is OPEN, create new bucket files if they do not exist. Otherwise, check the existing files for consistency.

Start the garbage collector thread

### 3.4.3.2 ~PersistentHashMap()

```
PersistentHashMap::~PersistentHashMap ()
```

Destroys the [PersistentHashMap](#) object.

Stop the garbage collector thread

## 3.4.4 Member Function Documentation

### 3.4.4.1 erase()

```
bool PersistentHashMap::erase (  
    std::string key)
```

Erases a key-value pair from the hash map.

## Parameters

<i>key</i>	The key of the key-value pair to erase.
------------	---

## Returns

True if the key-value pair was erased, false otherwise.

### 3.4.4.2 get()

```
std::optional< std::string > PersistentHashMap::get (  
    std::string key)
```

Retrieves the value of a key from the hash map.

**Parameters**

<i>key</i>	The key to retrieve the value for.
------------	------------------------------------

**Returns**

The value of the key, if it exists.

**3.4.4.3 insert()**

```
void PersistentHashMap::insert (
    std::string key,
    std::string value)
```

Inserts a key-value pair into the hash map.

**Parameters**

<i>key</i>	The key of the key-value pair.
<i>value</i>	The value of the key-value pair.

The documentation for this class was generated from the following files:

- include/persistent\_hashmap.h
- src/persistent\_hashmap.cpp

## 3.5 SpinLock Class Reference

A simple spin lock implementation using atomic\_flag.

```
#include <lru_cache.h>
```

**Public Member Functions**

- void **lock** ()  
*Locks the spin lock.*
- void **unlock** ()  
*Unlocks the spin lock.*

**3.5.1 Detailed Description**

A simple spin lock implementation using atomic\_flag.

This class provides a simple spin lock implementation using atomic\_flag.

The documentation for this class was generated from the following files:

- include/lru\_cache.h
- src/lru\_cache.cpp

## 3.6 StorageEngine Class Reference

A class to represent a disk-persistent key-value store.

```
#include <storage_engine.h>
```

### Public Types

- enum class [INITIALIZATION\\_MODE](#) { [CREATE](#) = 0 , [OPEN](#) = 1 }  
*An enumeration to specify the initialization mode of the database.*

### Public Member Functions

- [StorageEngine](#) (std::string directory, int num\_bins, [INITIALIZATION\\_MODE](#) mode=[INITIALIZATION\\_MODE::CREATE](#))  
*Constructs a new [StorageEngine](#) object.*
- [~StorageEngine](#) ()  
*Destroys the [StorageEngine](#) object.*
- void [insert](#) (std::string key, std::string value)  
*Inserts a key-value pair into the key-value store.*
- bool [erase](#) (std::string key)  
*Erases a key-value pair from the key-value store.*
- std::optional< std::string > [get](#) (std::string key)  
*Retrieves the value of a key from the key-value store.*

### 3.6.1 Detailed Description

A class to represent a disk-persistent key-value store.

This class provides a class to represent a disk-persistent key-value store. The key-value store stores key-value pairs in a persistent hash map. The class provides an interface to insert, erase, and get key-value pairs from the key-value store.

### 3.6.2 Member Enumeration Documentation

#### 3.6.2.1 INITIALIZATION\_MODE

```
enum class StorageEngine::INITIALIZATION\_MODE [strong]
```

An enumeration to specify the initialization mode of the database.

This enumeration specifies the initialization mode of the database. The database can be created or opened.

Enumerator

<a href="#">CREATE</a>	Create a new database.
<a href="#">OPEN</a>	Opens an existing database.

### 3.6.3 Constructor & Destructor Documentation

#### 3.6.3.1 StorageEngine()

```
StorageEngine::StorageEngine (
    std::string directory,
    int num_bins,
    INITIALIZATION\_MODE mode = INITIALIZATION\_MODE::CREATE)
```

Constructs a new [StorageEngine](#) object.

**Parameters**

<i>directory</i>	The directory where the key-value pairs are stored.
<i>num_bins</i>	The number of bins in the hash map.
<i>mode</i>	The initialization mode of the database.

### 3.6.4 Member Function Documentation

#### 3.6.4.1 erase()

```
bool StorageEngine::erase (  
    std::string key)
```

Erases a key-value pair from the key-value store.

**Parameters**

<i>key</i>	The key of the key-value pair to erase.
------------	---

**Returns**

True if the key-value pair was erased, false otherwise.

#### 3.6.4.2 get()

```
std::optional< std::string > StorageEngine::get (  
    std::string key)
```

Retrieves the value of a key from the key-value store.

**Parameters**

<i>key</i>	The key to retrieve the value for.
------------	------------------------------------

**Returns**

The value of the key, if it exists.

#### 3.6.4.3 insert()

```
void StorageEngine::insert (  
    std::string key,  
    std::string value)
```

Inserts a key-value pair into the key-value store.

## Parameters

<i>key</i>	The key of the key-value pair.
<i>value</i>	The value of the key-value pair.

The documentation for this class was generated from the following files:

- include/storage\_engine.h
- src/storage\_engine.cpp



# Chapter 4

## File Documentation

### 4.1 lru\_cache.h

```
00001 #pragma once
00002 #include <list>
00003 #include <unordered_map>
00004 #include <string>
00005 #include <atomic>
00006 #include <memory>
00007 #include <optional>
00008
00015 class SpinLock {
00019     std::atomic_flag flag {ATOMIC_FLAG_INIT};
00020
00021 public:
00025     void lock();
00026
00030     void unlock();
00031 };
00032
00039 struct LRUCacheElement {
00043     std::string key {};
00044
00048     std::string value {};
00049
00056     LRUCacheElement(std::string key, std::string value);
00057 };
00058
00065 class LRUCacheSegment {
00071     std::list<std::shared_ptr<LRUCacheElement> lruList {};
00072
00078     std::unordered_map<std::string, std::shared_ptr<LRUCacheElement> lruMap {};
00079
00085     int capacity {};
00086
00090     SpinLock lock {};
00091
00092 public:
00098     LRUCacheSegment(int capacity);
00099
00106     void put(std::string key, std::string value);
00107
00114     std::optional<std::string> get(std::string key);
00115
00121     void remove(std::string key);
00122 };
```

### 4.2 persistent\_hashmap.h

```
00001 #pragma once
00002 #include <lru_cache.h>
00003 #include <iostream>
00004 #include <fstream>
00005 #include <vector>
00006 #include <shared_mutex>
00007 #include <thread>
00008
```

```

00015 #ifdef LRU_CACHE_SIZE
00016 int const LRU_CACHE_CAPACITY = LRU_CACHE_SIZE;
00017 #else
00018 int const LRU_CACHE_CAPACITY = 64;
00019 #endif
00020
00026 #ifdef GC_INTERVAL
00027 int const GARBAGE_COLLECTOR_INTERVAL = GC_INTERVAL;
00028 #else
00029 int const GARBAGE_COLLECTOR_INTERVAL = 30;
00030 #endif
00031
00039 class BinControl {
00045     int bin_id {};
00046
00052     std::string bin_path {};
00053
00059     std::shared_mutex mutex {};
00060
00066     LRUCacheSegment cache {LRU_CACHE_CAPACITY};
00067
00068 public:
00076     BinControl(int bin_id, std::string bin_path, int cache_capacity);
00077
00084     void insert(std::string key, std::string value);
00085
00092     bool erase(std::string key);
00093
00100     std::optional<std::string> get(std::string key);
00101
00107     void compressFile();
00108
00114     void binCheck();
00115 };
00116
00124 class PersistentHashMap {
00130     std::string directory {};
00131
00135     int num_bins {};
00136
00142     std::vector<std::unique_ptr<BinControl> bin_controls {};
00143
00149     std::thread gc_thread {};
00150
00156     std::atomic<bool> stop_gc {false};
00157
00163     void runGarbageCollector();
00164
00171     int hash(std::string key) const;
00172
00173 public:
00180     enum class INITIALIZATION_MODE {
00184         CREATE = 0,
00185
00189         OPEN = 1
00190     };
00191
00199     PersistentHashMap(std::string directory, int num_bins, INITIALIZATION_MODE mode =
INITIALIZATION_MODE::CREATE);
00200
00204     ~PersistentHashMap();
00205
00212     void insert(std::string key, std::string value);
00213
00220     bool erase(std::string key);
00221
00228     std::optional<std::string> get(std::string key);
00229 };

```

## 4.3 storage\_engine.h

```

00001 #pragma once
00002 #include <string>
00003 #include <memory>
00004 #include <optional>
00005
00006 class PersistentHashMap;
00007
00015 class StorageEngine {
00016 public:
00023     enum class INITIALIZATION_MODE {
00027         CREATE = 0,
00028

```



```
00032         OPEN = 1
00033     };
00034
00042     StorageEngine(std::string directory, int num_bins, INITIALIZATION_MODE mode =
INITIALIZATION_MODE::CREATE);
00043
00047     ~StorageEngine();
00048
00055     void insert(std::string key, std::string value);
00056
00063     bool erase(std::string key);
00064
00071     std::optional<std::string> get(std::string key);
00072
00073 private:
00079     std::unique_ptr<PersistentHashMap> pImpl;
00080 };
```



# Index

- ~PersistentHashMap
  - PersistentHashMap, [13](#)
- binCheck
  - BinControl, [6](#)
- BinControl, [5](#)
  - binCheck, [6](#)
  - BinControl, [5](#)
  - compressFile, [6](#)
  - erase, [6](#)
  - get, [7](#)
  - insert, [7](#)
- compressFile
  - BinControl, [6](#)
- CREATE
  - PersistentHashMap, [11](#)
  - StorageEngine, [15](#)
- erase
  - BinControl, [6](#)
  - PersistentHashMap, [13](#)
  - StorageEngine, [16](#)
- get
  - BinControl, [7](#)
  - LRUCacheSegment, [9](#)
  - PersistentHashMap, [13](#)
  - StorageEngine, [16](#)
- include/lru\_cache.h, [19](#)
- include/persistent\_hashmap.h, [19](#)
- include/storage\_engine.h, [20](#)
- INITIALIZATION\_MODE
  - PersistentHashMap, [11](#)
  - StorageEngine, [15](#)
- insert
  - BinControl, [7](#)
  - PersistentHashMap, [14](#)
  - StorageEngine, [16](#)
- LRUCacheElement, [8](#)
  - LRUCacheElement, [8](#)
- LRUCacheSegment, [9](#)
  - get, [9](#)
  - LRUCacheSegment, [9](#)
  - put, [10](#)
  - remove, [10](#)
- OPEN
  - PersistentHashMap, [11](#)
- StorageEngine, [15](#)
- PersistentHashMap, [11](#)
  - ~PersistentHashMap, [13](#)
  - CREATE, [11](#)
  - erase, [13](#)
  - get, [13](#)
  - INITIALIZATION\_MODE, [11](#)
  - insert, [14](#)
  - OPEN, [11](#)
  - PersistentHashMap, [12](#)
- put
  - LRUCacheSegment, [10](#)
- remove
  - LRUCacheSegment, [10](#)
- SpinLock, [14](#)
- StorageEngine, [15](#)
  - CREATE, [15](#)
  - erase, [16](#)
  - get, [16](#)
  - INITIALIZATION\_MODE, [15](#)
  - insert, [16](#)
  - OPEN, [15](#)
  - StorageEngine, [15](#)